Campus-Wide Computing:
Early Results Using Legion at the University of Virginia[1]

Andrew S. Grimshaw
Anh Nguyen-Tuong
William A. Wulf

Abstract

The Legion project at the University of Virginia is an attempt to provide system services that provide the illusion of a single virtual machine to users, a virtual machine that provides *both* improved response time via parallel execution and greater throughput. Legion is targeted towards both workstation clusters and towards larger, wide-area, assemblies of workstations, supercomputers, and parallel supercomputers. Rather than construct Legion from scratch we are extending an existing object-oriented parallel processing system by aggressively incorporating lessons learned over twenty years by the heterogeneous distributed systems community. The campus-wide virtual computer is an early Legion prototype. In this paper we present challenges that had to be overcome to realize a working CWVC, as well as performance on a production biochemistry application.

1. Introduction

Providing resources to computationally demanding applications at the lowest cost is a challenge facing many organizations. The traditional solution to providing the necessary cycles has been to use a supercomputer. An alternative, less costly, solution that has emerged recently is to use networks of existing high-performance workstations instead, managing the collection of resources as a single entity. These systems are called variously “workstation farms” or “workstation clusters”. The advantage of the cluster approach is that the resources are often already in place, and under-utilized. A second advantage is that the cost per MIP/FLOP is much less. A key problem that must be addressed in cluster computing is management. The collection of workstations is just that, a collection. Without system software to tie the machines together it is not easy for a user to exploit cycles on many different workstations.

There are two broad categories of solutions to the problem of managing the workstation resources, throughput oriented systems, and response-time oriented systems. Throughput oriented systems are interested in exploiting available resources in order to service the largest number of *jobs*, where a job is a single program that does not communicate with other jobs. There are several

[1.] This work is partially funded by NSF grants ASC-9201822 and CDA-8922545-01, National Laboratory of Medicine grant (LM04969), NRaD contract N00014-94-1-0882, and ARPA grant J-FBI-93-116.

such systems available today, DQS, Condor, LoadLeveler, LSF, CODINE, and NQS to name a few [16][20][28][29][31][33]. Response time oriented systems are concerned with minimizing the execution time of a single application, i.e., with harnessing the available workstations to act as a virtual parallel machine. The purpose is to more quickly solve larger problems than would otherwise be possible on a single workstation. Examples of such parallel processing tools available for workstations include Express, Linda, Piranha, Mentat, P4, and PVM [9][24][7][46]. The two objectives, throughput and response time, are not necessarily mutually exclusive.

The problem with existing systems is that they are incomplete. One system may provide load balancing and a shared file space, but not parallel processing or fault-tolerance. Another system may provide high application performance via parallel execution yet provide no fault tolerance and require all hosts to see the same file system image. No single system currently provides all of the features required.

The Legion project at the University of Virginia is an attempt to provide system services that provide the illusion of a single virtual machine to users, a virtual machine that provides *both* improved response time via parallel execution and greater throughput [21]. Legion is targeted towards both workstation clusters and towards larger, wide-area, assemblies of workstations, supercomputers, and parallel supercomputers. Legion tackles problems not solved by existing workstation based parallel processing tools such as fault-tolerance, wide area network support, heterogeneity, the lack of a single file name space, protection and security, as well as providing efficient scheduling and comprehensive resource management. At the same time Legion provides the parallel processing, object-interoperability, task scheduling, security, and file system facilities not usually found in job-based load balancing systems.

Rather than attempt to construct Legion from scratch, we have chosen an evolutionary approach. We began by first constructing a campus-wide virtual computer (CWVC) testbed based on Mentat [24][25]. Mentat is an existing, robust, object-oriented, parallel processing system that supports execution across heterogeneous platforms. Starting with an existing system eliminates the long lead times and uncertainty associated with starting from scratch. Additionally, existing applications can be run in the new environment permitting ideas and system implementations to be tested using real rather than synthetic applications. For example, because Mentat is implemented

using replaceable modules (objects) we can easily experiment with new scheduling algorithms developed for the Legion environment without changing the applications.

The campus-wide virtual computer is a direct extension of Mentat onto a larger scale, and is a prototype for the nationwide system and reflects the fact that the university is a microcosm of the world. The computational resources at the University are operated by many different departments, there is no shared name space, and sharing of resources is currently rare. Resources owned by the departments consist of a variety of workstations (SUN, DEC, HP, IBM, SGI) and small parallel machines. This equipment is used for "production" applications during the day.

Even though the CWVC is much smaller, and the components much closer together, than in the envisioned nationwide Legion, it still presents many of the same challenges. The processors are heterogeneous, the interconnection network is irregular, with orders of magnitude differences in bandwidth and latency, and the machines are currently in use for on-site applications that must not be negatively impacted. Further, each department operates essentially as an island of service, with its own NFS mount structure, and trusting only machines in the island.

The CWVC is both a prototype and a demonstration project. The objectives are to:

- demonstrate the usefulness of network-based, heterogeneous, parallel processing to university computational science problems,
- provide a shared high-performance resource for university researchers,
- provide a given level of service (as measured by turn-around time) at reduced cost,
- act as a testbed for the nationwide Legion.

The prototype consists of over eighty workstations and an IBM SP-2 in six buildings and is now operational[2]. Before the system could become useful to users several challenges had to be overcome; simply scaling Mentat to run on the eighty machines was insufficient. The first challenge is to construct a federated file system and unified name space so that files, both executables and user data, can be accessed from any host in the system. The second challenge is to make the system able to deal with host and network failure in a graceful fashion. If one or more hosts fail the system must continue to operate without interruption. In a university environment, where some of the hosts are in public labs, periodic host failure is the rule and not the exception. The third challenge is protection and security. Mentat, like many other parallel processing systems,

[2] To see the CWVC in action connect to our web page at http://uvacs.cs.virginia.edu/~mentat/legion/ and follow the links to Cyberia. Cyberia is an on-line look at the running CWVC.

does not address security. Each user either starts their own copy of Mentat and runs in their protection domain. Alternatively, the user uses a shared copy (one set of daemons, always running), in which case their application runs in the same protection domain as the shared copy. Thus all of their data files must be world readable, and their output files must be world writable. This is acceptable for a small group of colleagues, but is unacceptable for a large system.

The last two challenges have their roots in human nature rather than technical necessity. The fourth challenge is to provide autonomy to workstation owners and system administrators. To allow users and local system administrators to control resource utilization on their hosts in order to enforce local scheduling policies and to allow users to throttle system utilization of their hosts. We call this “pain management” because without the ability to effect local utilization users feel that they are in pain and are reluctant to add their resources to the pool. The fifth and final challenge discussed here is resource accounting. To avoid the tragedy of the commons in which everyone uses resources yet none contribute it is necessary to know how much resource is contributed (offered and used) by each user and how much is consumed. Users who contribute more than they use can be rewarded, and those who consume more than the contribute can be charged. How users are rewarded and charged is a policy issue not yet addressed. Keeping track of resource utilization requires mechanism. Finally there is performance. While implementing solutions to the above challenges we must not let performance suffer! For many users that is after all the whole point.

In this paper we present our early results using the CWVC and outline the shape of the solutions to the six challenges. To demonstrate performance we use a biochemistry application, complib, that compares DNA and protein sequences. The performance results are encouraging. Several other production applications have been developed by and for university researchers, including planetary atmosphere simulations, steric acid circulation tank simulations, solid state circuit simulations, parallel genetic algorithms, and 3D image segmentation.

We begin our presentation with background material on Legion and Mentat. We then look at the CWVC, discussing progress over the last year in each of the six challenge areas. The computational environment is then described to provide context for the performance results. We then briefly describe complib, presenting an English description of the problem and its computational structure. The results are next, followed by a discussion of our next steps.

2. Background

2.1. Legion

Legion will consist of workstations, vector supercomputers, and parallel supercomputers connected by local area networks, enterprise-wide networks, and the National Information Infrastructure. The total computation power of such an assembly of machines is enormous, approaching a petaflop; this massive potential is, as yet, unrealized. These machines are currently tied together in a loose confederation of shared communication resources used primarily to support electronic mail, file transfer, and remote login. However, these resources could be used to provide far more than just communication services; they have the potential to provide a single, seamless, computational environment in which processor cycles, communication, and data are all shared, and in which the workstation across the continent is no less a resource than the one down the hall.

A Legion user has the illusion of a single, very powerful computer on her desk. It is Legion's responsibility to *transparently* schedule application components on processors, manage data transfer and coercion, and provide communication and synchronization in such a manner as to minimize execution time via parallel execution of the application components. System boundaries will be invisible, as will the location of data and the existence of faults.

Before the Legion vision can be realized, several technical challenges must be overcome. These are software problems; the hardware challenges are being addressed and are the enabling technologies that provide the opportunity. The software challenges revolve around eight central themes: *achieving high performance via parallelism, managing and exploiting component heterogeneity, resourc*e *management, file and data access, fault-tolerance, ease-of-use and user interfaces, protection and authentication, and exploitation of high-performance communications protocols*. We realize that these are serious issues; we examine them in more detail in [21][22].

In addition to the purely technical issues, there are also political, sociological, and economic ones. These include encouraging the participation of resource-rich centers and the avoidance of the human tendency to free-ride. We intend to discourage such practices by developing and employing accounting policies that encourage good community behavior.

2.2. Related Work

The vision of a seamless metasystem or metacomputer such as Legion is not novel. Indeed, a number of systems have been designed to attack one or more of the problems mentioned above, e.g., Andrew, Locus and NSF for file systems [32][35][49], Locus for fault-tolerance, Sun XDR and the University of Washington HCS for heterogeneity[39][40][45]. None has been fully successful. Several changes have occurred that make the realization of a complete high performance metacomputer possible. First, high-speed optical communication has revolutionized long distance data communication. Realized bandwidths have gone from T1 (1.5 mBits/Sec.) to OC/12 over Sonet (152 mBits/sec.), with similar magnitudes of change expected in the next five years. The increase in bandwidth makes nation-wide metasystems practical and not just a science fiction fantasy. The second change is that achieving high performance via parallelism, previously available only for tightly coupled parallel processors, is now possible for loosely coupled distributed systems [7][8][9][12][27][34][37][46][51]. Related advances in resource management driven by parallel computing have attacked the problem of decomposing and scheduling application components to minimize elapsed time.

Legion differs from other work in heterogeneous distributed systems [3][4][5][6][19][35][36][38][39][40][47][48][49] and object-oriented distributed systems [2][13][38]. One major difference is our emphasis on performance. Often interoperability, fault-tolerance, or consistency is the main focus, and performance is sacrificed. Legion though is performance oriented. The underlying model is parallel, and the user, not the system designers, will choose the appropriate level of fault tolerance and consistency that best meets their needs.

There are other metasystems [17][30][43][50] and heterogeneous parallel computing [7][10][18][46] projects underway. Our work differs from other heterogeneous parallel processing systems in the scope. We're addressing file systems, fault-tolerance, interoperability, etc. Our work differs from the other metasystems efforts in that we have a system already in place that can evolve - we are not starting from scratch.

As mentioned in the introduction several systems have been developed to manage workstation farms, DQS, Condor, LoadLeveler, LSF, and CODINE. Legion differs in that it is both a throughput and a response time system, we attempt to both better utilize system CPU resources

and decrease execution time via parallelism. Further, Legion provides a single name space to users with data accessible from anywhere in the system. In other words Legion is a complete system solution, rather than a load balancing tool.

Whether or not a metasystem is explicitly constructed by design, the nation (and perhaps the world) will eventually build a system that shares at least some of the attributes of Legion. The reason is simple: individual and organizational users will be required to deal with the increasingly obvious shortcomings of a computing infrastructure consisting of islands of computational power connected via the Internet. Internet tools such as *gopher*, *worldwide web* and *Mosaic* are examples of current attempts to bridge the gaps between local systems.

The issue is not whether metasystems will be developed; clearly they will. Rather, the question is whether they will come about by design and in a coherent, seamless system – or painfully and in an ad hoc manner by patching together congeries of independently developed systems, each with different objectives, design philosophies, and computation models.

2.3. Approach

The principles of the object-oriented paradigm are the foundation for the construction of Legion; our goal will be exploitation of the paradigm's encapsulation and inheritance properties, as well as benefits such as software reuse, fault containment, and reduction in complexity. The need for the paradigm is particularly acute in a system as large and complex as Legion. Other investigators have proposed constructing application libraries and applications for wide-area parallel processing using only low-level message passing services. Use of such tools requires the programmer to address the full complexity of the environment; the difficult problems of managing faults, scheduling, load balancing, etc., are likely to overwhelm all but the best programmers.

3. The CWVC

In order to achieve campus wide computing it is insufficient to simply run Mentat on all machines at the University. Mentat, like other existing parallel processing systems, was not designed to deal with system faults, non-overlapping file systems, and other problems that plague distributed systems as opposed to MPP's. Further, we do not believe that the Mentat programming language is the best language for all applications. Therefore, other models and languages will need

to be supported in the long run for Legion to be successful. Below we present the solutions we have implemented in the CWVC to the problems of files and I/O, fault tolerance, pain management, accounting, protection and security, and the need to support other programming models.

3.1. The Federated File System

A problem that arises when forming a system out of hosts in different organizations is that, unlike a single department, they usually do not share a single file system. This presents at least four difficulties; application binaries may not be present at all sites; application components may not be able to read and write files that they require for correct execution; sharing of data and results between collaborators at different sites requires either sending the files via email or copying them via ftp; and remote databases, e.g., genome databases, must be copied in their entirety into the local environment before they can be used, resulting in wasteful copies, out-of-date data, and the inconvenience of frequently manually copying the data.

An obvious solution to this problem is to extend some existing file system such as AFS or NFS to both the campus wide and nation wide system. There are problems with this approach. First, some file systems such as NFS simply will not scale to the level we require. They require far too much human intervention (setting up mount points, etc.). Further, the use of NFS requires that all users have the same user id on all hosts, a requirement we simply cannot meet. The Andrew file system, on the other hand, is scalable. The problem with using Andrew is the requirement that all hosts run Andrew. We feel that we cannot impose a file system standard on participating organizations. Further, we do not feel that the system should dictate a particular file semantics as Andrew does; file semantics should be based on the file type. Nevertheless we intend to borrow heavily from early distributed file system projects such as Andrew because issues such as naming, location transparency, fault transparency, replication transparency, and migration have been addressed both in the literature [32] and in one or more existing operational systems.

Our solution is to construct a federated file system using the local host file systems as the component elements. The basic idea is simple and consists of two parts, constructing a Legion name space on top of the existing file system, and a modified set of *stdio* libraries that interact with the Legion file space and Legion file objects. The Legion name space is implemented using a

collection of name servers that point to instances of file classes. A file is in the Legion name space if it's name (complete path name) is found in one of the name servers. Otherwise it is a local file.

The name servers keep track of the name of the file, e.g., "/legion/grimshaw/mywork1", and a set of file attributes[3]. The attributes include the class of the file, e.g., unix_file, restrictions on the placement of class instances, e.g. on hosts with a ".cs.virginia.edu" suffix, and an initialization string to be passed to an object instance on instantiation. This string is most often the Unix path name of the actual file data.

We also provide a set of library routines that support the *stdio* interface, e.g., open, close, read, write, etc. These library routines are linked by applications and intercept calls to the I/O system. All open calls are first examined to determine whether they are Legion files or local files. Local file operations are passed onto the host operating system. Legion file operations are trapped and passed onto the unix_file objects. This technique of trapping file operations is not unique to Legion and was first used in Unix United. Unfortunately the relinking technique does not work on

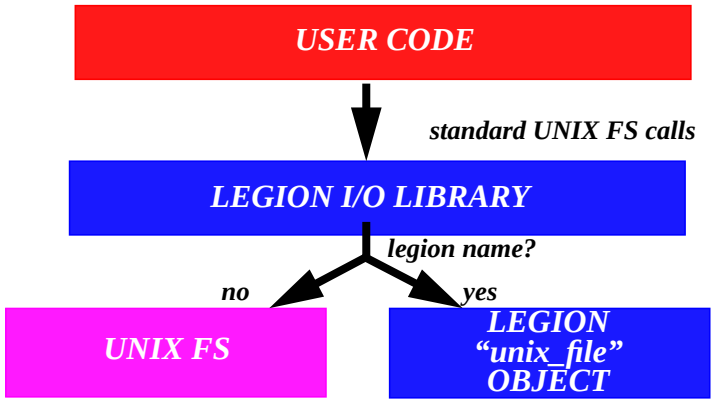


Figure 1 Emulation of the *stdio* interface allows applications to use the Legion federated file system capabilities without modification.

all supported platforms. Therefore we have had to design and implement a separate I/O library for use by applications on those platforms.

Finally, there exist a set of operations to manipulate the Legion namespace that are analogous to operations that manipulate the Unix name space, e.g., lmkdir, lrm, lmv, etc. Files enter the Legion namespace via one of three mechanisms, they are created in the Legion name space,

3. The implementation details are changing–the philosophy is not.

they are moved into the Legion name space, or they are linked into the Legion names space (similar to a soft-link).

3.2. System Fault Tolerance

With a system as large as the CWVC it is a certainty that a machine will go down every few hours. Further, whole buildings will lose power, while other buildings will not. Mentat, and other network-capable parallel processing systems were not designed to address any form of fault tolerance. We knew that if we wanted to construct a facility that others would use it had to be highly available (there when they needed it) and robust to failures of all kinds, host failure, network failure, application failure, etc. The mechanism to provide full application fault-tolerance is a long term goal and is still in the research and design stage. In the short term though, the system itself must be fault tolerant or no one will use the it; contemporary computer users have very high expectations with regard to system availability. Before we can describe the current fault tolerance mechanism a brief bit of background is required.

Each host in a running Legion system has at least two daemons (exclusive of the fault tolerance daemons). The first is the instantiation manager (IM) which is responsible for scheduling, keeping track of all Legion objects on the host, monitoring the load on the system, and other management functions. The second is the token matching unit (TMU) which supports language features [26]. The system configuration file (config.db) contains the names of all hosts in Legion[4]. The set of IM's running on those hosts define the system. If an IM goes down then Legion is down on that host. Further, user programs, user objects, and IM's communicate with IM's in order to carry out their functions. If an IM on a host goes down (either because of a program fault or host failure) objects on other hosts will not receive a response and will lock up, i.e., they will fail. To eliminate this form of failure we must ensure either that hosts and IM's never fail, an impossible goal, or deal with failure when it occurs.

To deal with failure we have constructed a new class of daemons, phoenix, that monitors the system for failure. Phoenix is responsible for restarting system components if possible. If they cannot be restarted phoenix notifies the remaining IM's of a configuration change. The IM's in turn notify all objects on their host. If a host has been removed from the configuration due to failure

[4] The configuration database is a non-scalable entity. Future releases will use a different mechanism.

phoenix will begin to monitor the host for recovery. When the host has recovered, phoenix will restart Legion on that host and notify the other IM's of the configuration change.

The phoenix instances are arranged in a tree structure. The *root* phoenix starts *cluster* phoenix instances on a single host in each cluster. These cluster phoenix instances in turn start a *leaf* phoenix instance on each host in the cluster. The leaf phoenix instances monitor the health of the local daemons, restarting them as necessary. The root phoenix monitors the health of the leaves. When one fails (usually due to a host failure) the root restarts it. This design is non-scalable and has a single point of failure (the root). Because of the single point of failure we place the root on our most reliable host. A long term solution to both the stability and single point of failure problems is to use a distributed algorithm. So far though, neither has been a serious problem.

Transparent application fault-tolerance is being investigated but is not currently implemented. We believe that fault-tolerance services are a necessary condition for success of any large scale system such as Legion. We expect to have fault-tolerance for stateless objects and applications which use them by Supercomputing. This encompasses a large class of applications.

3.3. Pain Management

A third problem is pain. The pain that workstation "owners" feel when a Legion job is executing on their machine. Because of the pain, and the need not to antagonize resource owners, we have constructed a Legion "thermostat" (Figure 2) that permits resource owners to set maximum resource use limits (in percent CPU and physical memory) on Legion programs running on their hosts. The thermostat mechanism works in a fashion similar to many home thermostats. The owner can select resource availability during different time intervals. Further, just as with a home thermostat, if the owner is unhappy with the current setting, she may change it at any time. (Just like a home thermostat it may take a few minutes before the load is adjusted to the new level.)

The system guarantees that Legion resource consumption will stay below the limits imposed by the user. The available resources are divided between user objects running on the host. Thus, some user objects will be "throttled", resulting in possibly longer execution times for applications. We feel that the trade-off between autonomy and application performance must be made in favor of the local user. If it is not, then resource owners may withdraw their resources.

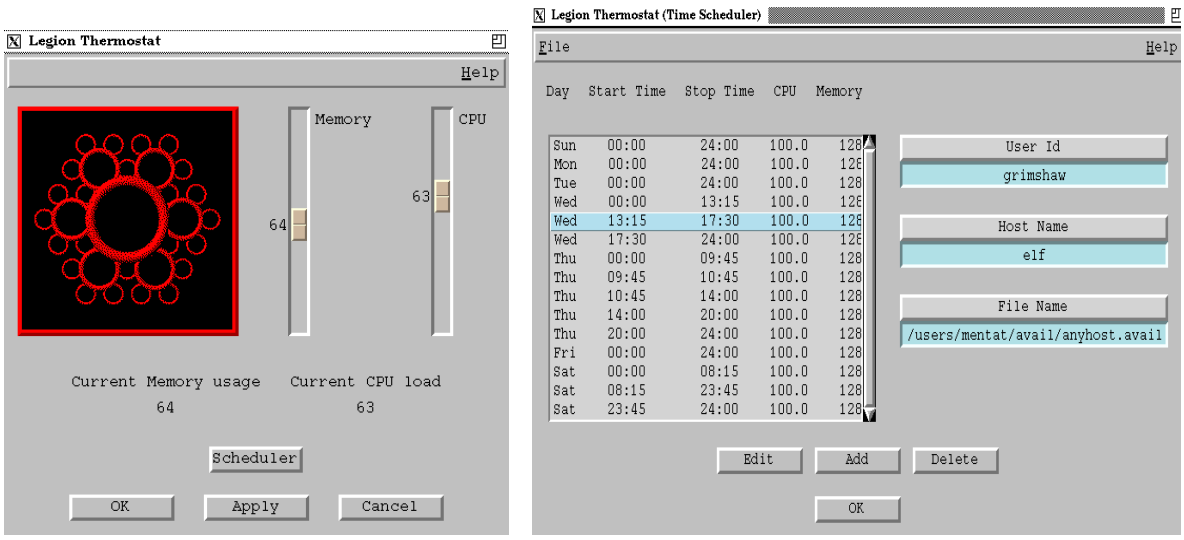


Figure 2 Legion Thermostat. Authorized users can specify both a resource schedule and modify the current resource limits. The interface on the left controls current resource limits. By changing the sliders resource limits may be temporarily modified. The "scheduler" button brings up the interface on the right, which allows authorized users to change the daily schedule.

In order to encourage resource owners to participate and specify high resource limits the resource owner receives credits when resources are consumed and when resources are offered *even if they are not used*. The amount of credit received depends on the type of system, the time of day, and how much resource was offered and used. Different amounts of credit are received for offered but not used, and offered and used resource. The scale factors (for type of system and time of day) are in essence prices. Prices are a policy issue determined by system administrators.

3.4. Accounting

If Legion and the CWVC are successful then all a user will need in order to have access to the set of resources managed by the system is an interface such as an X-Windows interface. In such an environment a rational user will purchase just the interface, and leave the purchase of expensive resources to other users. It would not be long before no new equipment was purchased, resulting in an impoverished system. To avoid this classic "tragedy of the commons", mechanism and policy are required to encourage good community behavior. In the case of Legion and the CWVC this takes the form of an accounting system which monitors resource contribution and consumption.

Accounting for resource consumption is accomplished by associating each instantiated user object with an owner, a CWVC user, and monitoring user object activity. The owner-id is the Unix UID of the shell which launched the application[5]. Resource consumption is monitored on an object-by-object basis. The run-time libraries used by objects are instrumented to collect information such as the amount of CPU used, the number of messages sent and received, the total volume of data moved in and out of the object, and the number of method invocations performed. On object termination this information is forwarded to the instantiation manager. The instantiation manager in turn places the resource data into a host specific accounting database. Periodically the accounting data is collected, merged into a single database, and resource reports are generated.

Resource contribution is monitored similarly. Actual resource contribution is derived from the resource consumption database; if resources were used on a host by an object then they were contributed by that host. Offered resources are determined by maintaining thermostat logs. Recall the thermostat is used to throttle resource consumption on a host by restricting usage to a particular percentage. The thermostat log indicates when thermostat settings were changed on a host.

Both resource contribution and consumption are scaled using the resource scale file. The scale file consists of scale factors for each host type, the time of day, and whether the resource was consumed, offered, or contributed. This is used to account for the fact that, for example, a Sparc IPC CPU second at 1:00 AM is not worth nearly as much as an SGI CPU second at noon. The scale factors can be thought of as prices. Using the accounting files and the resource scale file we can calculate a resource balance for each user. The resource balance is the users amount of surplus or deficit. A system-wide surplus, available to system administrators, can be generated by maintaining a spread between the price for resources consumed and resources offered.

The mechanism lets us to know who is using and contributing how much resource. Without a policy on resource consumption collecting the information is an exercise in programming only. Policy is still being worked out with resource holders. We envision a situation in which users with chronic resource deficits can either buy (using dollars) more resource or receive “grants” from the system. The grants will come out of the system-wide surplus. The funds generated can either be used to pay users with chronic surpluses, or be re-invested in additional equipment.

---

[5] Use of the Unix ID is temporary, they are not consistent across systems. A Legion user ID, LUID, will be used in the future, introducing authentication issues not yet addressed in the implementation.

3.5. Protection and Security

Protection and security are critical in wide-area distributed systems. Mentat does not address security; instead it relies on having read/write permission to all application databases. Our security plan has long-term and short-term components.

Long term. Legion cannot solve the general security problem, but a reasonable level of privacy and integrity of data must be provided or the system will not be used. Legion will run on top of whatever operating system is available on the participating host, thus at first blush it may seem that the issue of security would be determined by the lowest common denominator of those host systems. The first rule of Legion security, like the Hippocratic Oath, will be to "do no harm." In the first instance, that means that there should be no possibility of host data being compromised by a Legion task. In the second instance it means that no Legion user's data should be compromised by a rogue host. The mechanisms are reasonably well understood and include:

- execution of Legion tasks with "least privilege" on the host,
- storing no persistent objects on a "foreign host",
- using cryptographic authentication protocols to verify that a server is valid and that a service request originated with a valid Legion task, and
- digital signatures to validate that data has not been corrupted.

Further, when needed for greater security, more costly techniques can be used, for example:

- messages can be encrypted,
- scheduling can be restricted to certain classes of nodes, and
- white noise can be injected into the communication stream.

Another goal of Legion security is to provide a better model and mechanism for Legion-to-Legion security than is provided by the underlying hosts. The object-oriented computational model suggests a capability-based access control model in which the "rights" are type (class) specific and map one-to-one onto the methods of the class. The distributed and heterogeneous nature of the system suggests the use of encrypted capabilities to ensure that they cannot be forged.

Short term. In the short-term we have implemented a security scheme which a determined attacker could compromise. The basic idea is simple. Users "transfer" files to the Legion name space. The file is placed in a hidden directory that only they have read privilege to, and a soft-link is created with the same name as the old file. This ensures that the user can still modify the file. At the same time a hard-link is created for Legion to the file. This permits Legion applications to read

and write the file. The net result is that both the user and Legion programs may read and write the file, but other users cannot. The “hole” in this scheme is that other users Legion programs may read and write the file.

3.6. Support for other Models - PVM V 3.0

Clearly we cannot expect all Legion users to use MPL. Other parallel processing languages and models must be supported. This is particularly true for legacy codes. Our plan for legacy code and multiple model interoperability is detailed in [21]. One of the early tests for our plan is support for PVM [46]. PVM is a message passing parallel processing system that has gained wide-spread acceptance in the user community, and is a *de facto* standard.

At first glance it seams counter-intuitive to implement PVM on top of a system such as the CWVC and Legion; they are usually layered on top of systems like PVM. We have implemented PVM on the CWVC so that existing PVM applications can be executed in the CWVC environment without the need to re-write the application. The ported PVM codes will not only execute, they will benefit from CWVC features such as the federated file system, pain management, and our load-sensitive scheduling algorithms. This ability to use existing codes, and to later integrate them with other parallel codes, support our larger Legion goals of legacy code support and interoperability.

The implementation is straightforward. Each PVM instance is represented by a typed Mentat object. Calls such as *pvm_initiate*(), *pvm_send()*, and *pvm_recv*() are implemented using calls to the CWVC run-time system that instantiate objects, send messages, receive messages, etc. PVM-specific services, e.g., barriers, are implemented using synchronization objects.

To both test the correctness of our implementation, and to compare the performance of our implementation against the “native” PVM implementation, we took to already implemented PVM applications and ran them in the same environment. The two applications are the latency/ bandwidth test code that comes with the PVM distribution, and the PVM implementation of the NAS benchmarks. TABLE 1 presents the performance comparison between the native PVM and Mentat-PVM implementations on the communication benchmarks. One can see that for short messages (<= 10,000 bytes) the Mentat-PVM implementation outperforms the native implementation. The Mentat-PVM time is the result of the optimized communication system used

TABLE 1 Point to Point benchmarks on 40 mhz Sparc 2's[a].

Message size (bytes)	Native PVM time (mSec)	Mentat-PVM time (mSec)	Native PVM bandwidth (bytes/Sec)	Mentat-PVM Bandwidth (bytes/sec)
100	8.81	8.47	11,800	13,310
1000	9.88	9.75	101,269	102,550
10,000	27.52	26.58	363,380	376,220
100,000	194,231	196,783	514,860	509,880
1,000,000	1,920,624	1,951,618	520,670	512,790

a. All measurements are on an 8 processor cluster using raw (no XDR) encoding, and direct routing.The time is the time to send a message of the specified size and to receive a one byte reply. It is the average of 3 runs (each run being the average of 20 sends/receives).

by Mentat. That short messages are faster is significant. Most messages in many applications are short messages.

To test application performance we selected the PVM NAS benchmark implementation because it is familiar to most readers. TABLE 2 presents the performance results. The measurements were taken using eight, 40 mhz Sparc 2's, using raw encoding.

TABLE 2 NAS Benchmark Results

	Time (Sec)		Communication Time (Sec)		Communication Volume (MB)	
Application	**Native PVM**	**Mentat-PVM**	**Native PVM**	**Mentat-PVM**	**Native PVM**	**Mentat-PVM**
IS Kernel[a]	226	256	202	207	140	140
EP Kernel[b]	346	350	NA	NA	NA	NA
MG Kernel[c]	123	110	62	59	49	49

a. 2^{21} keys in the range $[0..2^{19}]$
b. reduced problem size of 2^{26}
c. problem size of 128

4. Results

4.1. The Application - Protein and DNA Sequence Library Comparison

Our test application, *complib*, compares two protein or DNA sequence libraries. Each library contains of one or more sequences, each of which consists of a sequence name and a variable length string of characters (also known as residues) that represent the sequence. Each sequence in the first library, called the source library, is compared against each sequence in the second, target, library. For each sequence comparison a score is generated reflecting sequence

commonality using one of several algorithms. Three popular algorithms are Smith-Waterman [44], FASTA [39], and Blast [1]. The latter two algorithms are heuristics; the quality of the score is traded for speed. Smith-Waterman is the benchmark algorithm, generating the most reliable scores although at considerable time expense. We used the Smith-Waterman algorithm to compare our performance against previously published numbers in the biochemistry community [14][42].

Once all of the scores have been generated for one source sequence against all target sequences the scores are sorted and statistical information is generated. Thus, there are two phases that are executed sequentially for each source sequence, sequence comparison, and data reduction. An important attribute of the comparison algorithms is that all comparisons are independent of one another and, if many sequences are to be compared, they can be compared in any order. This natural data-parallelism is easy to exploit.

The main program manipulates three objects, the source genome library, the target genome library, and a recorder object that performs the statistical analysis and saves the results. The application is written in the Mentat programming language and is described in detail in [23]. The main program for loop is shown in Figure 3 below. The effect is that a pipe is formed, with sequence extraction from the source, sequence comparison in the target, and statistics generation are executed in a pipelined fashion. Each high-level sequence comparison is transparently expanded into a fan-out, fan-in program graph where the “leaves” are the workers, the source sequence is transmitted from the root of the tree to the leaves, and the results are collected and sorted by collators.

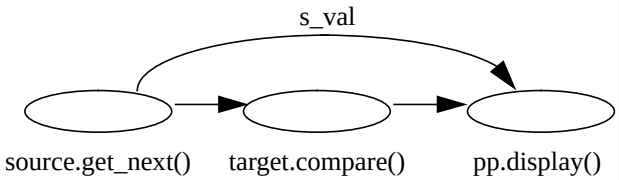
4.2. The Campus Computing Environment

All tests were conducted on the grounds-wide[6] network at the University of Virginia. The available computing resources are shown in TABLE 3. The hosts were physically located in six different buildings. Each building has one or more Ethernet segments connected to the grounds-wide fiber backbone by routers. Some of the hosts are several “hops” from the fiber backbone. For these experiments all machines had at least one shared NFS mount point where all executables and data were stored. In general this is not a satisfactory solution as we cannot count on cross-mounted file systems.

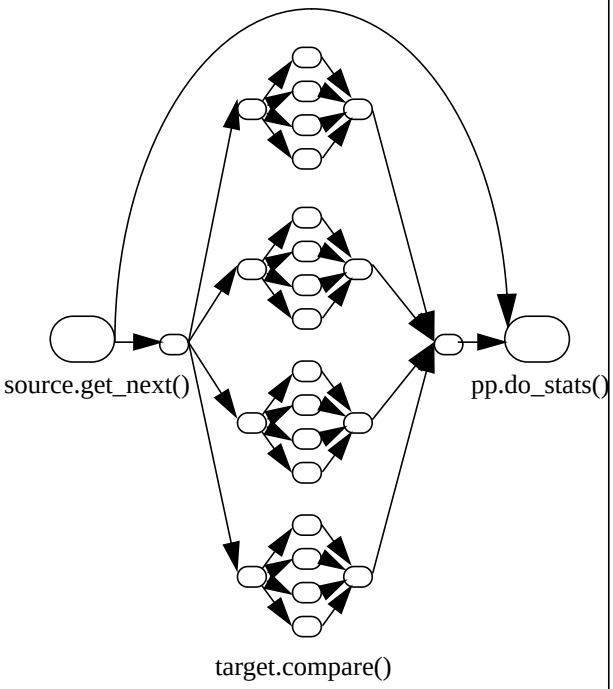
6. At the University of Virginia, the campus is called the “grounds”.

```
for(i=0;i<num_source_seq;i++) {
    //for each sequence
    s_val = source.get_next();
    //Compare against target library
    result = target.compare(s_val);
    //Do statistics
    post_process.do_stats(result,s_val);
}
```

(a)



(b)



(c)

Figure 3 Mentat implementation of *complib*. The main loop of the program is shown in (a). Three objects are manipulated, the source, the target, and the post_processor. The pipelined program graph is shown in (b). *Target.compare()* has been expanded showing sixteen workers in (c). The fan-out tree distributes the source sequence to the workers. The internal nodes of the reduction tree are collator objects. The reduction tree sorts and merges the results generated by the workers.

4.3. Complib results & discussion

All measurements were performed in early January 1995 during the Winter break. In general, workstations were operating under light loads except for two of the departmental Sparc 10 compute servers. Execution times for complib were generated using a 20 sequence source library containing 4478 residues and a 10,716 sequence target library consisting of 3,647,403 residues. We follow the tradition of the biochemistry community of reporting performance numbers in terms of millions of matrix entries per second. The number of matrix entries is obtained by multiplying the number of residues in the source library with the target library. In our experiment, there were 4478*3,647,403 residues, or approximately 16,333 million matrix entries.

To provide a benchmark for comparison we executed a sequential version of complib on all the platforms that comprise the CWVC. TABLE 3 lists the sequential execution times as well as the corresponding million matrix entries per second. In each case, the user CPU time was reported.

TABLE 3 . Sequential complib(20 vs. 10716 sequences, 16,333 million matrix entries)

Platforms	Number	Sequential Time (sec)	Matrix entries per second (millions)
Sparc 10 (fast)	4	13460	1.21
Sparc 10 (slow)	8	13998	1.17
Sparc LX	6	23146	0.71
Sparc 2	5	33823	0.48
Sparc IPC	41	57259	0.29
SGI Indigo	17	11386	1.43

TABLE 4 gives the parallel execution times on the CWVC as well as the speedup relative to each of the platforms. The number of workers ranges from 8 to 64. Three runs of each were performed. The times shown represent the best wall clock time obtained excluding the initial overhead of distributing the target library[7]. All times are given in seconds.

TABLE 4 . CWVC Times & Relative Speedups

		Relative speedup					
Workers	**CWVC**	**SGI**	**Sparc 10 (fast)**	**Sparc 10 (slow)**	**Sparc LX**	**Sparc 2**	**Sparc IPC**
8	1692	6.7	7.9	8.3	13.7	20.0	33.8
16	926	12.3	14.5	15.1	25.0	36.5	61.8
24	951	12.0	15.2	14.7	24.3	35.6	60.2
32	724	15.7	18.6	19.3	32.0	46.7	79.1
64	592	19.2	22.7	23.6	39.1	57.1	96.7

[7]. In table 6, we show the performance of complib on the CWVC against other platforms[14][42]. All measurements reported were obtained by excluding the initial overhead time.

4.4. Is speedup meaningful in the CWVC?

A generalized definition for speedup in a heterogeneous environment is given in [15]. In our particular case, we characterize the performance of complib in terms of the number of matrix entries per second obtained. Furthermore, we define the efficiency of running complib on the CWVC as:

$$efficiency\,(\text{CWVC},\text{complib}) \;=\; \frac{\text{measured performance on the CWVC}}{\text{maximum theoretical performance}}$$

The maximum theoretical performance is obtained by assuming that workers are placed on the more powerful processors first. For example, in the case of 24 workers, the theoretical maximum performance is 32.66 million matrix entries per second (17 workers placed on the sgis, 4 on the faster sparc10s, and 3 on the slower sparc10s). Note that efficiency is dependent on both the particular application, and the types and numbers of hosts that make up the CWVC.

TABLE 5 . Efficiency

Workers	Theoretical maximum matrix entries per second (millions)	Matrix entries per second (millions) on the CWVC	Efficiency
8	11.44	9.65	.84
16	22.88	17.64	.77
24	32.66	17.17	.54
32	40.64	22.56	.55
64	52.13	27.59	.53

The efficiency obtained for 24 to 64 workers is slightly above 50%. There are two causes for the reduced efficiency: other users competing for cycles, and a poor initial partition of the data.Our initial partitioner did not handle the heterogeneous processor case well. We have re-implemented the partitioner and expect better results in the future.

While efficiency gives us a measure of the overhead in running complib on the CWVC, the important performance number for biochemists is the number of matrix entries per second. We compare the number of matrix entries per second obtained against other platforms in TABLE 6. In terms of performance, the CWVC is roughly equivalent to the 32 node Paragon at the Jet Propulsion Lab.

TABLE 6 . Comparison against various platforms[a]

Platform	Matrix entries per second (millions)
CWVC	27.59
CM-2 (32000 proc)	65
Paragon (32 nodes)	29
5 DEC Alpha AXP 300	18

a. The CM-2 performance numbers were obtained from [14], the Paragon and DEC Alphas' from [42].

4.5. Problems encountered

We encountered several problems transforming Mentat into the CWVC. Some of these, such as the need for a federated file system and single name space were anticipated. Others were not. At least three of these problems are not Legion specific, they will need to be overcome by most systems that have the same objectives as Legion/CWVC.

The first problem is the trend in workstation operating systems towards the exclusive use of dynamic linkers. For example, Irix, Solaris, and AIX do not support static linking. This is a problem whenever hosts do not share the same file system structure, in other words object libraries may be at a different location on different hosts, or not present at all. Thus, an executable that executes on one host, will not execute on another host of the same architecture type. This limits our ability to transport binaries from one location to another. Executable transport is not an issue in older operating systems such as SunOS which permit static linking.

Another facet of this problem occurred when we implemented the stdio library call traps.We could not statically link our routines in to replace the underlying C library routines. To solve this problem required that we explicitly manage the dynamic linker in our code. Unfortunately the mechanisms required vary from system to system.

A second class of problems that we encountered relate to Unix itself. It is not sufficient to simply scale the number of hosts when using a Unix based parallel processing tool if there is a single host that starts remote shells executing on other hosts. For example, if a daemon on host A starts daemons on hosts B..Z. This technique, which we used to practice, and which PVM uses begins to fail when there are many hosts for at least two reasons. First, the number of open files can rapidly exceed the limits of the operating system. (These limits can be changed by recompiling the

operating system.) The problem is that each rsh consumes at least two file descriptors, and possibly more if the pipe() command is used as well. Even if these file descriptors are closed by the "main" daemon they are not always closed immediately as called for in the manuals. Instead, there is usually a timeout of just under five minutes. Under normal operating conditions this is not a problem, but when a very large number of rsh's are being generated in a short period of time, such as when starting the parallel system up, the system may run out of descriptors.

A related problem has to do with the use of NFS file servers. Simultaneous execution of a large number of copies (> 60) of the same executable, as in a data parallel program or system start-up, may overload the file server, resulting in multiple lost requests. The host operating systems treat this as an error, and report that the executable does not exist, when clearly it does.

A final problem we encountered is application fault-tolerance. While the system itself is fault-tolerant and recovers from host failure, applications do not. If my application has an object on a host that has failed, then my application blocks, and never recovers. This requires the user to kill and restart the application. We consider this unacceptable in the long run. We have implemented fault-tolerance in two applications at the application level, and are exploring general application fault-tolerance.

5. Summary and On-going Work

Legion is an ambitious project to construct a nation-wide virtual computer. Rather than attempting to construct Legion from scratch we have chosen to begin with an existing system, Mentat, and transform it into Legion by incorporating research results from over twenty years of heterogeneous distributed computing. The first step in the transformation is the construction of the campus-wide virtual computer at the University of Virginia. The objectives of the campus-wide virtual computer are to:

- demonstrate the usefulness of network-based, heterogeneous parallel processing to university computational science problems,
- provide a shared high-performance resource for university researchers,
- provide a given level of service (as measured by turn-around time) at reduced cost,
- act as a testbed for the nationwide Legion.

The prototype implementation is well on its way to meeting these objectives. The performance results provide evidence that workstation based, heterogeneous parallel processing can be used to solve computationally challenging problems of interest to university researchers at

reduced cost. With respect to the testbed goal, the CWVC has been an invaluable tool which has enabled us to begin trying out designs and stressing implementations with real applications. Our experience over the last year putting the CWVC together highlighted several critical factors that must be addressed in both the short term and in the long term.

Finally, Legion and the CWVC are not static; both are works in progress. We will continue to enhance the system and extend the applications running on the CWVC. With respect to Legion we are forging ahead with enhanced security, protection, resource management, fault-tolerance, I/O, and naming services. In addition we are working on an OS/2 Warp port.

6. Acknowledgments

We would like to thank Bill Pearson of the biochemistry department for introducing us to sequence comparison, and for collaborating on the parallel version. We would also like to thank all of the members of the Legion team. The faculty are Bill Wulf, Jim French, Paul Reynolds Jr., and Alf Weaver. Staff members are Mark Hyett and Lindsey Faunt. The students who have worked on various components are Jon Weissman and Anh Nguyen-Tuong (run-time system), John Karpovich, Matt Judd, and Adam Ferrari (federated file system and I/O libraries), Emily West (complib) and with Mark Morgan (pain management and thermostat), Chenxi Wang (security), Roger Harper (PVM), and Mike Lewis. We would also like to thank the Jet Propulsion Lab for the use of their Intel Paragon.

7. References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, “Basic local alignment search tool”, *J. Mol. Biol.*, 215, pp. 403-410, 1990.

[2] H. Bal, J. Steiner, and A. Tanenbaum, “Programming Languages for Distributed Computing Systems,” *ACM Computing Surveys*, pp. 261-322, vol. 21, no. 3, Sept. 1989.

[3] A. Black, N. Hutchinson, E. Jul, and H. Levy, ``Distribution and Abstract Types in Emerald,” University of Washington, TR 85-08-05, August, 1985.

[4] G. Bernard et al., “Primitives for Distributed Computing in a Heterogeneous Local Area Network Environment”, *IEEE Trans on Soft. Eng*. vol. 15, no. 12, December 89.

[5] B. N. Bershad, and H. M. Levy, “Remote Computation in a Heterogeneous Environment.” Tech. Rep. 87-06-04, Dept. of Computer Science, University of Washington, Seattle, June, 1987.

[6] B. N. Bershad, et al., “A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems,” *IEEE Trans. Software. Eng. SE*, vol. 13, no. 8, pp. 880-894, August, 1987.

[7] J. Boyle et al., *Portable Programs for Parallel Processors*, Holt, Rinehart and Winston, New York, 1987.

[8] D. Callahan and K. Kennedy, “Compiling Programs for Distributed-Memory Multiprocessors” *The Journal of Supercomputing*, no. 2, pp. 151-169, 1988, Kluwer Academic Publishers.

[9] N. Carriero and D. Gelernter, “Linda in Context,” *Communications of the ACM*, vol. 32, no. 4, pp. 444-458, April, 1989.

[10] N. Carriero, D. Gelernter, and T.G. Mattson, “Linda in Heterogeneous Computing Environments,” *Proceedings of WHP 92 Workshop on Heterogeneous Processing*, IEEE Press, pp. 43-46, March, 1992.

[11] B. Chapman, P. Mehrotra, and H. Zima, “Programming in Vienna Fortran,” Scientific Programming, vol. 1, no. 1, Aug. 1992, pp. 31-50.

[12] A.L.Cheung, and A.P. Reeves, “High Performance Computing on a Cluster of Workstations,” *Proceedings of the First Symposium on High-Performance Distributed Computing*, pp. 152-160, Sept., 1992.

[13] R. Chin and S. Chanson, “Distributed Object-Based Programming Systems,” *ACM Computing Surveys*, pp. 91-127, vol. 23, no. 1, March., 1991.

[14] A. S. Deshpande, D. S. Richards, and W. R. Pearson, “A platform for biological sequence comparison of parallel computers”, *CABIOS*, 7, pp. 237-247, 1991.

[15] V. Donaldson, F. Berman, and R. Paturi, “Program Speedup in a Heterogeneous Computing Network,” *Journal of Parallel and Distributed Computing*, vol. 21, pp. 316-322, 1994.

[16] F. Ferstl, “CODINE Technical Overview,” Genias, April, 1993.

[17] R. F. Freund and D. S. Cornwell, “Superconcurrency: A form of distributed heterogeneous supercomputing,” Supercomputing Review, Vol. 3, Oct. 1990, pp. 47-50.

[18] E. Gabber, “VMMP: A Practical Tool for the Development of Portable and Efficient Programs for Multiprocessors”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, July 1990.

[19] P. B. Gibbond, “A Stub Generator for Multi-Language RPC in Heterogeneous Environments,” *IEEE Trans. Software. Eng. SE*, vol. 13, no. 1, pp. 77-87, January, 1987.

[20] T. P. Green and J. Snyder, “DQS, A Distributed Queueing System,” Florida State University, March 1993.

[21] A. S. Grimshaw, W. A. Wulf, J. C. French, A.C. Weaver, and Paul Reynolds Jr. “Legion: The Next Logical Step Toward a Nationwide Virtual Computer,” Computer Science Technical Report, University of Virginia, CS 94-21, June, 1994.

[22] A.S. Grimshaw, J.B.Weissman, E.A. West, and E. Loyot, “Meta Systems: An Approach Combining Parallel Processing And Heterogeneous Distributed Computing Systems,” *Journal of Parallel and Distributed Computing*, pp. 257-270, vol. 21, no. 3, June, 1994.

[23] A. S. Grimshaw, E. A. West, and W.R. Pearson, “No Pain and Gain! - Experiences with Mentat on Biological Application,” *Concurrency: Practice & Experience*, pp. 309-328, Vol. 5, issue 4, July, 1993.

[24] A. S. Grimshaw, “Easy to Use Object-Oriented Parallel Programming with Mentat,” *IEEE Computer*, pp. 39-51, May, 1993.

[25] A. S. Grimshaw, W. T. Strayer, and P. Narayan, “Dynamic Object-Oriented Parallel Processing,” *IEEE Parallel & Distributed Technology: Systems & Applications*, pp. 33-47, May, 1993.

[26] A. S. Grimshaw, J. B. Weissman, and W. T. Strayer, “Portable Run-Time Support for Dynamic Object-Oriented Parallel Processing,” *to appear in ACM TOCS*, and earlier version is available in Computer Science Technical Report, CS-93-40, University of Virginia, July, 1993.

[27] P. J. Hatcher, et al, “Data-Parallel Programming on MIMD Computers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 3, pp. 377-383.

[28] International Business Machines Corporation, “IBM LoadLeveler: User’s Guide.”, Kingston, NY, March 1993.

[29] J.A. Kaplan and M.L. Nelson, “A Comparison of Queueing, Cluster, and Distributed Computing Systems,” NASA Technical Memorandum 109025, NASA LaRC, October, 1993.

[30] Ashfaq Khokhar, et. al., “Heterogeneous Supercomputing: Problems and Issues,” *Proceedings of WHP 92 Workshop on Heterogeneous Processing*, IEEE Press, pp. 3-12, Beverly Hills, CA, March, 1992.

[31] B. A. Kingsbury, “The Network Queueing System,” Palo Alto, CA, March 1993.

[32] E. Levy, and A. Silberschatz, “Distributed File Systems: Concepts and Examples,” ACM Computing Surveys, vol. 22, No. 4, pp. 321-374, December, 1090.

[33] M. Litzkow and M. Livny, “Experience with the Condor Distributed Batch System”, *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, October, 1990.

[34] D. B. Loveman, “High Performance Fortran,” *IEEE Parallel & Distributed Technology: Systems & Applica-*

tions, vol. 1, no. 1, pp. 25-42, February, 1993.

[35] J.H. Morris, et al., 'Andrew: A distributed personal computing environment', *Communications of the ACM*, vol. 29, no. 3, March 1986.

[36] S. Mullender ed., *Distributed Systems*, ACM Press, 1989.

[37] N. Nedeljkovic, and M.J. Quinn, "Data-Parallel Programming on a Network of Heterogeneous Workstations," *Proceedings of the First Symposium on High-Performance Distributed Computing*, pp. 28-36, Syracuse, NY, Sept., 1992.

[38] J.R. Nicol, C.T. Wilkes, and F.A. Manola, "Object-Orientation in Heterogeneous Distributed Systems", *IEEE Computer*, vol.26, no. 6., pp. 57-67, June, 1993.

[39] D. Notkin, N., et al.,"Heterogeneous Computing Environments: Report on the ACM SIGOPS Workshop on Accommodating Heterogeneity," *Communications of the ACM*, vol. 30, no. 2, pp. 132-140, February, 1987.

[40] D. Notkin, et al., "Interconnecting Heterogeneous Computer Systems," *Communications of the ACM*, vol. 31, no. 3, pp. 258-273, March, 1988.

[41] W. R. Pearson and D. Lipman, "Improved tools for biological sequence analysis", *Proc. Natl. Acad. Sci. USA*, 85, pp. 2444-2448, 1988.

[42] W. R. Pearson, *Personal communication*, February, 1995.

[43] R. Rouselle et al., "The Virtual Computing Environment," *Proceedings of the Third International Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, August, 1994.

[44] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences", *J. Mol. Biol.*, 147, pp. 195-197, 1981.

[45] Sun Microsystems. *External Data Representation Reference Manual.* Sun Microsystems, Jan. 1985.

[46] V.S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, vol. 2(4), pp. 315-339, December, 1990.

[47] R. N. Taylor, et al., "Foundations for the Arcadia Environment Architecture", *Proceedings of the Third ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development*,

[48] M.M. Theimer, and B. Hayes, "Heterogeneous Process Migration by Recompilation," *Proc. 11th Intl. Conference on Distributed Computing Systems*, Arlington, TX, May, 1991, pp. 18-25.

[49] B. Walker, et al., "The LOCUS Distributed Operating System," *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, N. H., Oct.) ACM, New York, 1983

[50] Mu-Cheng Wang, et. al., "Augmenting the Optimal Selection Theory for Superconcurrency," *Proceedings of WHP 92 Workshop on Heterogeneous Processing*, IEEE Press, pp. 13-22, Beverly Hills, CA, March, 1992

[51] Min-You Wu, and G.C. Fox, "A Test Suite Approach for Fortran90D Compilers on MIMD Distributed Memory Parallel Computers," *Proceedings of the First Symposium on High-Performance Distributed Computing*, pp. 393-400, Syracuse, NY, Sept., 1992.